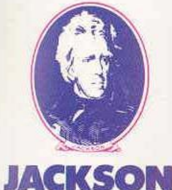


20 LECCIONES DE BASIC
PARA APRENDER CON EL SPECTRUM

INGELEK



JACKSON

Los ports de comunicaciones

Conexiones serie y paralelo

Operadores lógicos, tablas lógicas de verdad

STOP, CONTINUE, CLEAR, NEW

Objetivo de programación: síntesis y claridad

Videoejercicios

Videojuego N.º 7

7

Spectrum

16K/48K/PLUS

VIDEO BASIC
Una publicación de
INGELEK JACKSON
Director editor por INGELEK:
Antonio M. Ferrer
Director editor por JACKSON HISPANIA:
Lorenzo Bertagnolio
Director de producción:
Vicente Robles
Autor: Softidea
Redacción software italiano:
Francesço Franceschini,
Stefano Cremonesi
Redacción software castellano:
Fernando López, Antonio Carvajal,
Alberto Caffarato, Pilar Manzanera
Diseño gráfico:
Studio Nuovaidea
Ilustraciones:
Cinzia Ferrari, Silvano Scolari,
Equipo Galata
Ediciones INGELEK, S. A.
Dirección, redacción y administración,
números atrasados y suscripciones:
Avda. Alfonso XIII, 141
28016 Madrid. Tel. 2505820
Fotocomposición: Espacio y Punto, S. A.
Imprime: Gráficas Reunidas, S. A.

ISBN del tomo 2: 84-85831-17-9
ISBN del fascículo: 84-85831-11-X
ISBN de la obra completa: 84-85831-10-1
Depósito Legal: M-15076-1985
Plan general de la obra:
20 fascículos y 20 casetes, de aparición quincenal,
coleccionables en 5 estuches.
Distribución en España:
COEDIS, S. A.
Valencia, 245. 08007 Barcelona.
INGELEK JACKSON garantiza la publicación de todos
los fascículos y casetes que componen esta obra y el
suministro de cualquier número atrasado o estuche
mientras dure la publicación y hasta un año después de
terminada.
El editor se reserva el derecho de modificar
el precio de venta del fascículo,
en el transcurso de la obra, si las circunstancias del
mercado así lo exigen.
Julio, 1985.
Impreso en España.



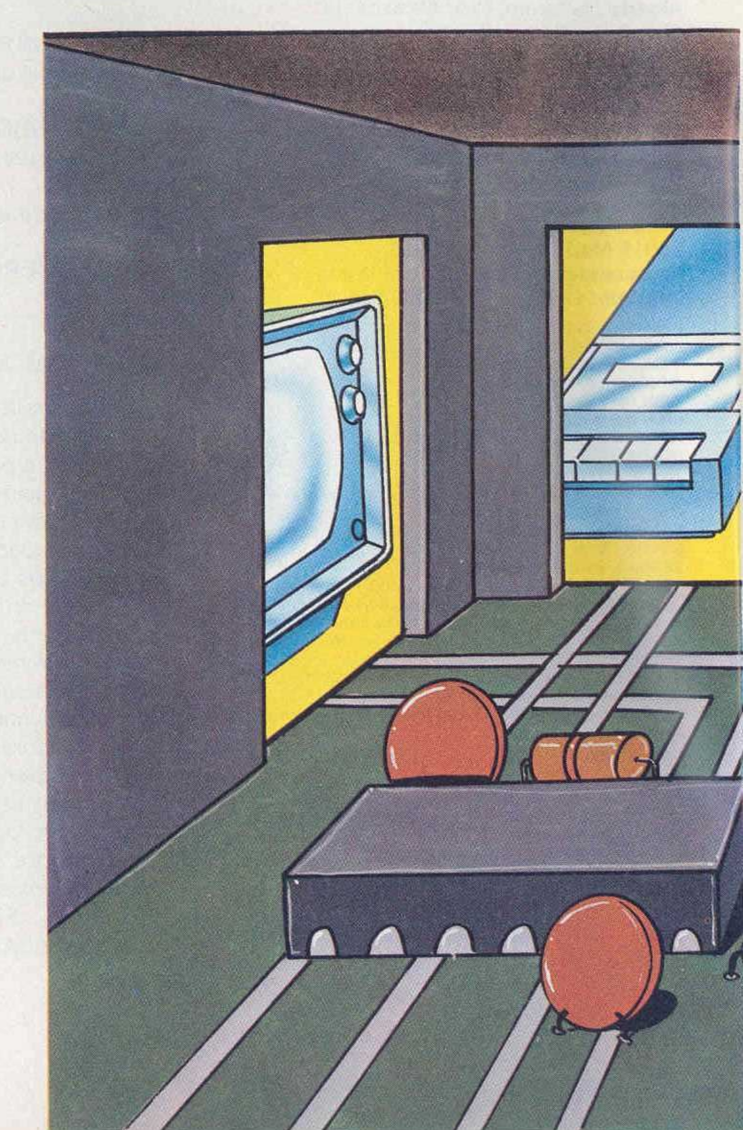
SUMARIO

Introducción

Existen dos técnicas para la transmisión de datos: en serie y en paralelo. La primera es más sencilla y barata, la segunda más rápida. En tu C64 ambas coexisten para permitir las comunicaciones entre ordenador y periféricos y entre las diferentes partes del ordenador mismo.
De otra parte, ya habíamos indicado que el ordenador es capaz de tomar «casi decisiones»; seguiremos pues por este camino introduciendo ahora los operadores lógicos (AND, OR, NOT) que permiten comparaciones y decisiones, según esquemas y estructuras lógicas semejantes a las seguidas por el hombre en sus razonamientos.
Y además... STOP, CONTINUE, NEW, BREAK, CLEAR.

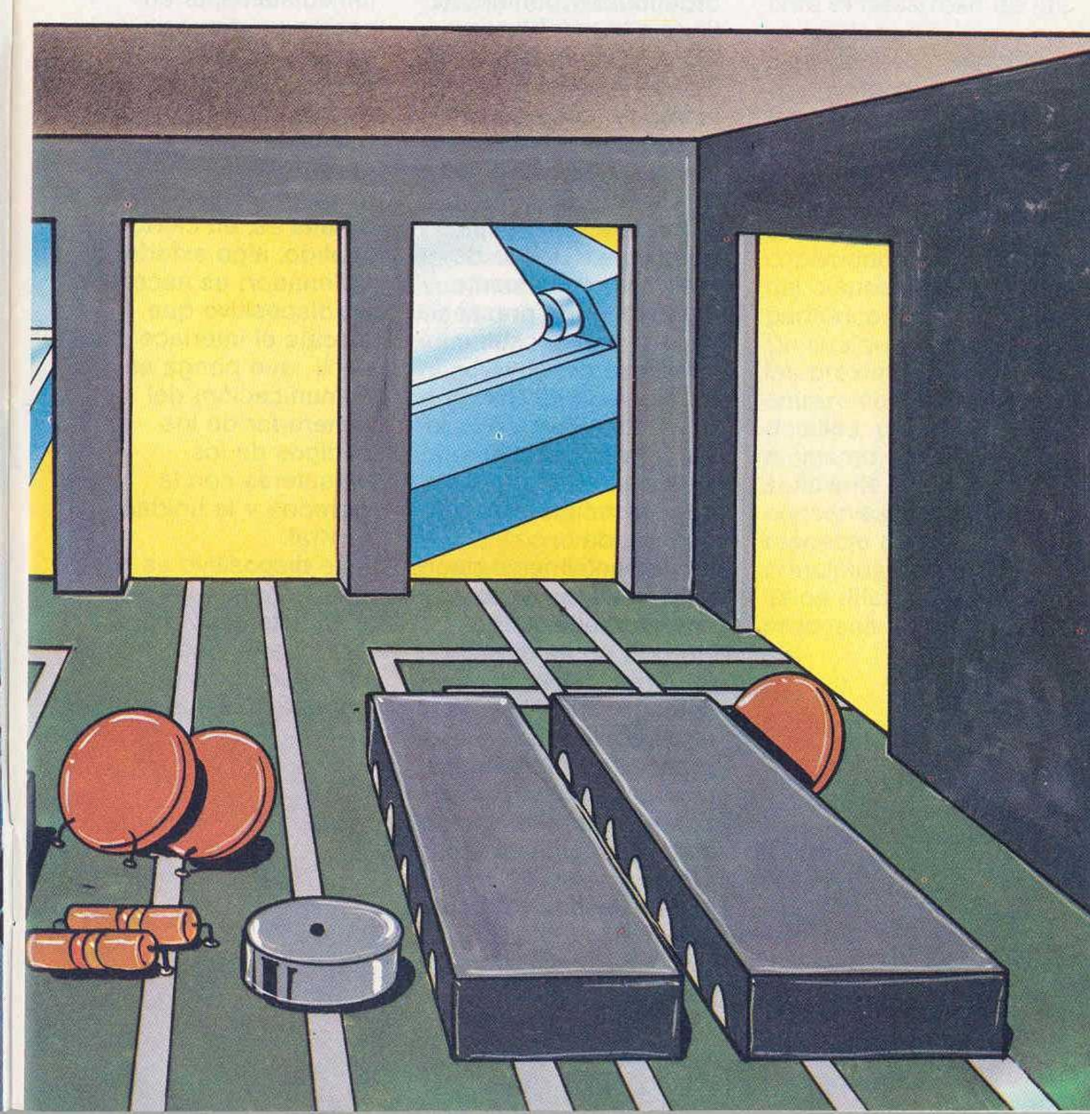
Los ports

Ya hemos visto en las lecciones pasadas (¡y lo volveremos a ver en las siguientes!) cómo el corazón del ordenador, es decir, la CPU, es capaz de efectuar –a traves de largas pero rapidísimas operaciones elementales– todas las elaboraciones, los cálculos y las ordenaciones necesarias para llevar a cabo las diferentes secuencias de acciones especificadas en cada caso en el programa existente en memoria. Elaborar datos mediante ordenador significa precisamente obtener los resultados necesarios a través de las citadas series de operaciones.
Pero lo que quizá no haya quedado aún suficientemente explicado es la importancia fundamental que revisten en todo el



proceso las distintas unidades periféricas. Cada cálculo comporta un conjunto de operaciones (de lectura, localización, elaboración, transferencia y emisión), tan múltiple y complejo, que ninguna unidad central por sí sola, es decir, sin los adecuados y necesarios soportes ejecutivos, puede ser capaz de afrontarlos y resolverlos.

Como nos proponemos examinar durante las lecciones las descripciones específicas y detalladas de las diferentes unidades periféricas, tenemos que ocuparnos ahora de un tema habitualmente



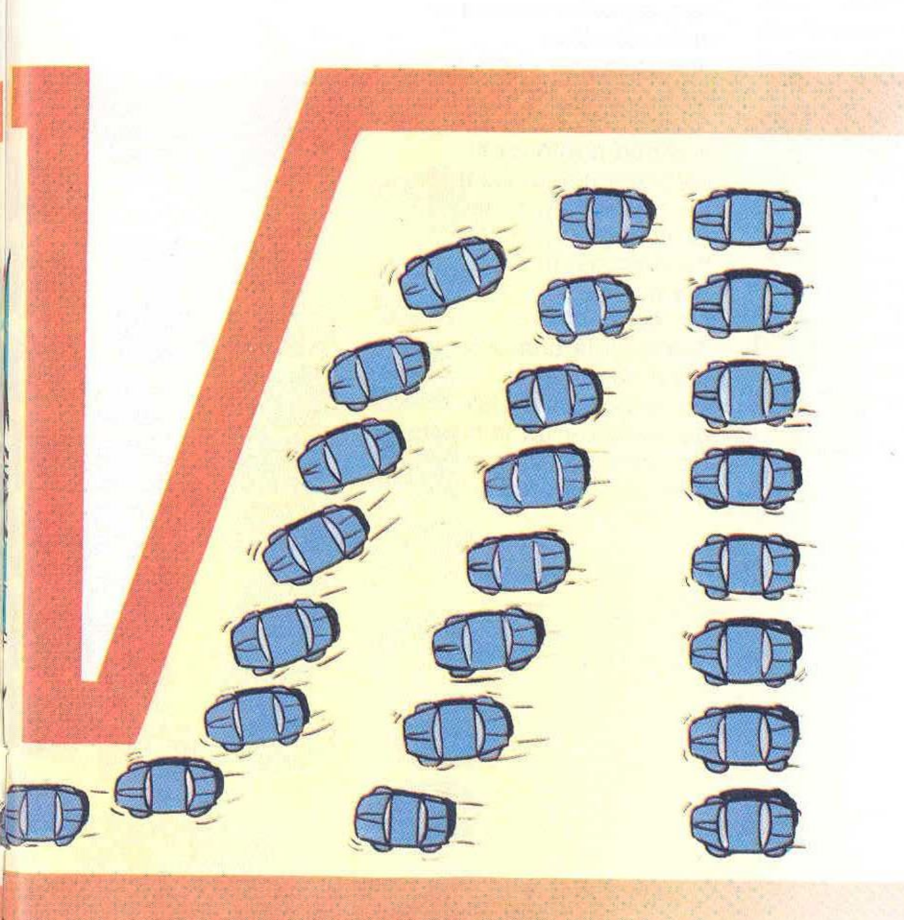
infravalorado o hasta ignorado pero no por ello de importancia secundaria, sino todo los contrario: las técnicas de conexión y comunicación entre la unidad central y los periféricos, o, en terminos más generales, los interfaces entre la unidad central y el mundo exterior.
Está claro, o debiera estarlo, que cualquier «coloquio» entre ordenador y periféricos tiene que establecerse —con independencia del tipo de periférico— a través de una conexión eléctrica.
Por lo tanto, habremos de ver en primer lugar el modo en que tales conexiones deben de realizarse físicamente.
Excluyendo la presencia de uno o más cables conductores, cuya función es unir los diferentes elementos, la parte principal del sistema de comunicación está constituida fundamentalmente por los llamados «ports» de entrada/salida.
Un port no es más que un circuito eléctrico (situado normalmente en el interior del ordenador) capaz, mediante unas condiciones adecuadas y prefijadas, de dejar entrar o salir del ordenador las informaciones que puedan ir siendo necesarias y estén disponibles.
Un ejemplo te lo aclarará inmediatamente.
Cuando pulsas una tecla cualquiera de tu ordenador, la presión que has ejercido sobre el teclado pone inmediatamente en acción un dispositivo que, como bien sabes, genera la combinación de bits correspondiente a la tecla pulsada.
Pero hasta este momento la secuencia de bits es, en cierto sentido, algo exterior al ordenador; es necesario un dispositivo que efectúe el interface (es decir, que ponga en comunicación) del generador de los códigos de los caracteres con la memoria y la unidad central.
Este dispositivo es precisamente un port y en este caso un port de entrada, dado que la información es de entrada. Gracias a ellos,

el carácter pulsado logra «entrar» en el ordenador, dispuesto para disposición de las posibles elaboraciones.
Todo parece sencillo: cuando la CPU desea realizar una operación de entrada o salida, le es suficiente con leer o escribir (mediante los procedimientos oportunos) el dato a intercambiar a través del port correspondiente al periférico seleccionado, y el problema está resuelto.
Sin embargo, las cosas no son así de sencillas. La velocidad de trabajo de la unidad central es muy diferente a la de cualquier periférico. Si los datos a intercambiar son más de uno (y esto es lo que ocurre en la inmensa mayoría de los casos), puede ocurrir que la CPU efectúe la totalidad de sus operaciones a una velocidad tan elevada que vuelve a intentar leer o escribir en un port antes de que el periférico haya tenido tiempo de «digerir» el dato anterior.
En el caso de una instrucción de entrada, este hecho provoca una lectura doble (o triple o cuádruple...) del mismo dato, mientras que en el caso de una instrucción de salida le cambia el «sentido» de las operaciones a un periférico en medio de una operación, con resultados lógicamente imprevisibles.
Se hace, por lo tanto, indispensable un «depósito» intermedio, es decir, un elemento en el que las informaciones se puedan acumular sin mayores consecuencias para el desarrollo de las operaciones.
Este depósito, llamado «buffer» o memoria tampón, cumple la tarea de «absorber» los datos a alta velocidad, pasándolos después uno tras otro según la capacidad para tratarlos del dispositivo periférico.
De cualquier manera, en las próximas lecciones entraremos en más detalles; por el momento, será suficiente con que conserves ideas claras respecto de los principios básicos de estos diferentes procesos.

Transmisión en serie y en paralelo

Hablaremos ahora de la realización de la conexión entre ordenador y periférico. Es posible dividir las técnicas de conexión en dos categorías generales: serie y paralelo.
En la conexión en serie, los bits que componen cada dato (¿recuerdas? 8 bits = 1 byte) se envían, uno detrás del otro, y bajo forma de impulsos eléctricos, por el mismo cable y a intervalos regulares de tiempo.
Por lo tanto, por el cable pasan unos «trenes» de impulsos,

debidamente distanciados entre sí, cada uno de los cuales es precedido por una señal de «llegada de dato» que para el circuito receptor tiene el significado de que se prepare para la llegada del dato.
Es un tipo de conexión económica y sencilla de realizar, y en consecuencia, es de uso frecuente en los ordenadores personales.
A cambio, no permite alcanzar velocidades elevadas de transmisión, ni realizar conexiones muy largas; como máximo unos diez metros.
En la mayoría de los casos, estos defectos, si es que se puede hablar de defectos, resultan prácticamente irrelevantes para el empleo de un ordenador personal.
En cambio, para la

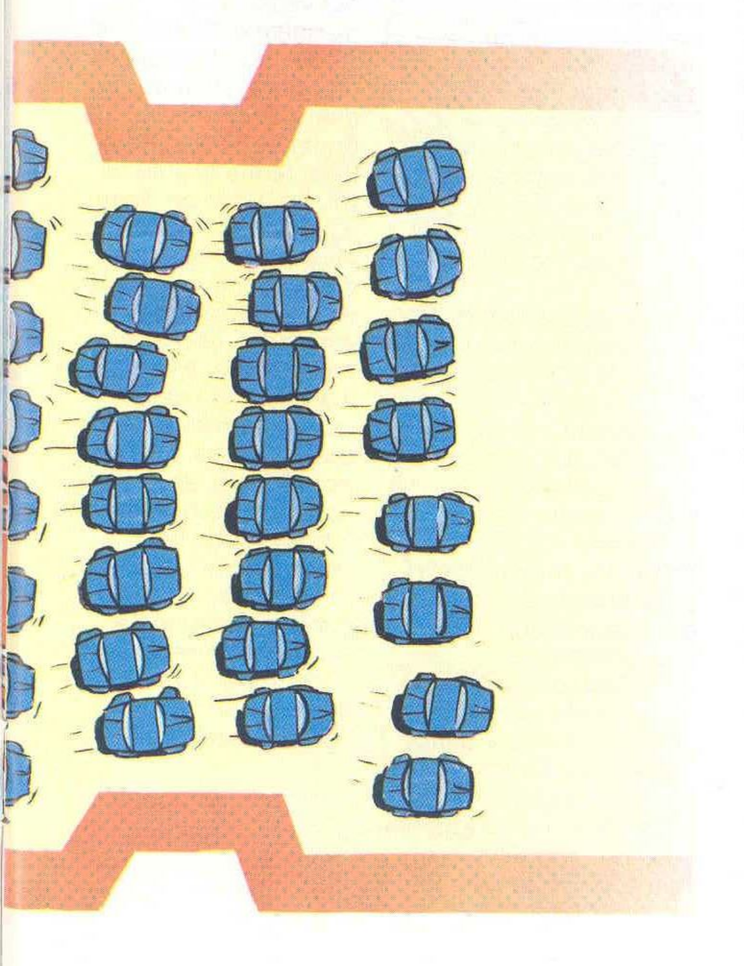


conexión en paralelo se emplea un hilo por cada bit a transmitir; así el cable está constituido por 8 hilos, más los necesarios para usos auxiliares y comunicaciones de servicio, sobre las que no nos extenderemos más.

Ahora, los bits que componen los datos a transmitir ya no se envían uno detrás del otro, sino uno al lado del otro. Es como si en una autopista de 8 carriles, cada bit recorriera uno de ellos reservado exclusivamente para sí.
El defecto principal de esta solución salta a la vista: se eleva notablemente el coste de la conexión.
A pesar de esto, muchos periféricos emplean como sistema de comunicación la conexión en paralelo.
Pasaremos ahora a las normas de construcción.
Tanto en la conexión en serie como en la paralela, con el objeto de asegurar un mínimo de compatibilidad entre dispositivos producidos por casas distintas, se han desarrollado protocolos de conexión y de comunicación, es decir, normas, que establecen cómo tienen que conectarse los hilos, qué características deben cumplir las señales eléctricas, en qué orden



y a qué velocidad deben enviarse las señales, etc.
Una de las normas para conexión más extendida (más aún, prácticamente el único en el campo de los ordenadores pequeños), es la versión simplificada del protocolo RS 232 C/V 24, llamado popularmente RS 232.
Un comentario a propósito de velocidades de comunicación: éstas especifican el número de bits por segundo que se pueden enviar a través de la línea.



La unidad de medida es pues el bit por segundo (bps), llamado también tasa de baudios. Son valores típicos en baudios de la velocidad de transmisión de datos aquellos comprendidos entre los 40 y los 19200.
Como habrás podido ver, el interface entre una unidad central y un periférico no es un tema sencillo y de solución rápida. Son necesarios estudios en profundidad sobre la compatibilidad de los diversos elementos, sobre las normas y protocolos de comunicación y sobre los procedimientos de construcción.
En próximas lecciones, cuando hablemos detalladamente de cada periférico, volveremos con más detalle sobre el tema.

Los operadores lógicos

Ya hemos comentado otras veces que la gran difusión que han alcanzado los ordenadores, depende en gran parte de su extremada velocidad y precisión en la realización de cálculos y comparaciones.
Y son especialmente estas últimas las que permiten a los ordenadores aproximarse y emular unas pautas de acción que le resultan al hombre sencillas y familiares, puesto que emplea la comparación, es decir, un razonamiento, según esquemas y estructuras lógicas.
Así, en este aspecto, los hombres y las máquinas están de acuerdo: aquello que es normalmente más sencillo para el hombre resulta serlo también para la máquinas, y viceversa.
Todos los lenguajes de programación cuentan entre sus instrucciones más potentes e importantes con aquella que permite efectuar comparaciones y tomar decisiones. Gracias a élla es posible programar siguiendo el mismo modelo lógico (basado normalmente en una serie de exclusiones mutuas) que es aquél que en el fondo cualquier ser humano acostumbra a seguir cada vez que se encuentra ante la obligación de decidir entre un determinado conjunto de posibilidades y elecciones.
Por ejemplo, la decisión derivada de un razonamiento como el siguiente: «mañana iré a esquiar si hace buen día, si Pedro viene conmigo, y si consigo levantarme pronto. Si no, iré a dar un buen paseo», no podrá alcanzarse más que después de que hayan sido alcanzados y evaluados uno a uno todos los factores sobre los que se basa el razonamiento, y de los que depende.
Por lo tanto, es necesario estructurar en forma similar las instrucciones de los programas correspondientes a tales decisiones.
Pero hasta ahora todas las expresiones condicionales (o

relacionales) que hemos encontrado eran de un tipo sencillo: podían operar sobre los elementos de la condición (o de la relación) únicamente mediante una distribucción de tipo verdadero-falso, si-no, mayor-menor, uno-cero. Por lo tanto, es necesario para poder ejecutar desde el interior de las expresiones condicionales y lógicas todas las operaciones que puedan ser útiles o necesarias, disponer de una nueva «familia» de operadores: los llamados operadores lógicos.



Hasta ahora habíamos conocido y empleado únicamente operadores aritméticos (con los que elaborar operaciones matemáticas) y operadores relacionales (para efectuar comparaciones). Los operadores lógicos en cambio, se usan para combinar en un único resultado lógico, dos valores lógicos distintos.

Los operadores lógicos son:

AND (el uno y el otro)
OR (uno u otro)
NOT (negación)

Una comparación del tipo: 5 > 3 es verdadera. Por convenio, a la expresión 5 > 3 se le asigna el valor 1.

LENGUAJE

La condición $8 < 5$ no se verifica, es decir, es falsa. Por tanto, a la expresión $8 < 5$ le corresponde (siempre por convención) el valor 0.
Gracias a este hecho es posible escribir expresiones de este tipo:

```
LET A = (5 > B)
```

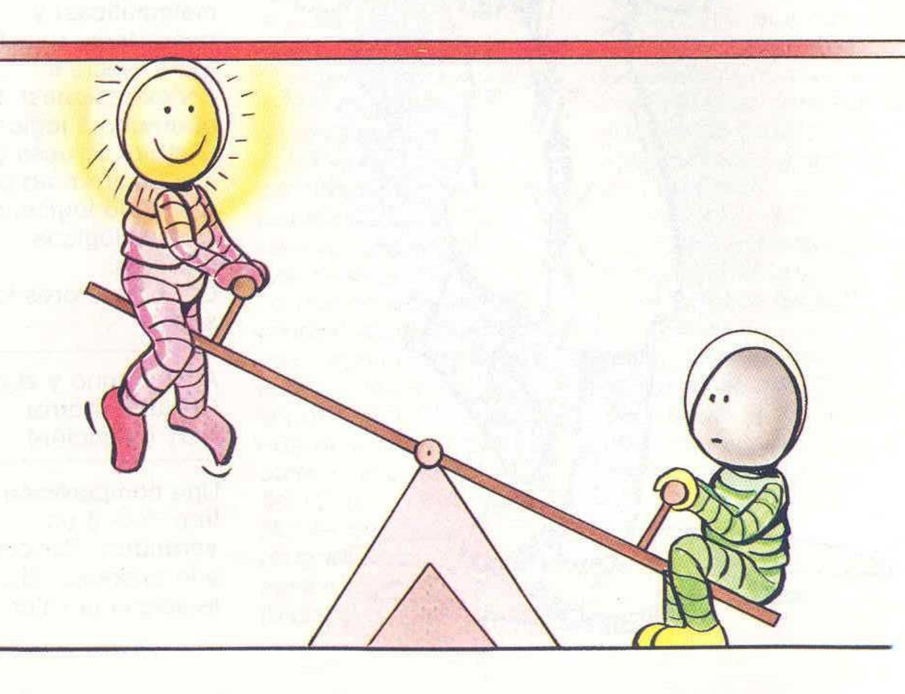
A la variable A se le asigna el valor 0 ó 1, dependiendo del hecho de que B resulte mayor o menor que 5:

```
LET K = (4 = A)
```

K valdrá 0 ó 1 dependiendo de que A resulte distinto o igual a 4.

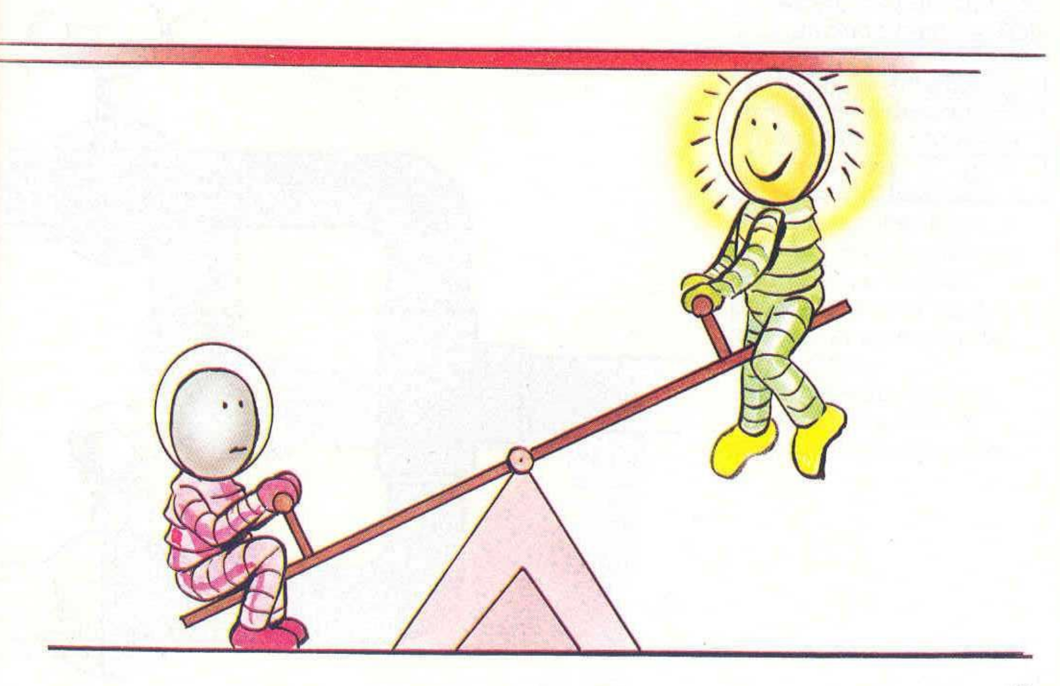
NOT

NOT es la operación lógica más elemental; significa negación, es decir, inversión del estado de una variable o de una relación.
Así, si A es una variable con valor 0, su negación –NOT A– vale 1. Si en cambio, A posee cualquier valor distinto de 0, NOT A vale 0.



Ejemplos:

`LET S = NOT 5`	Asigna a S el valor 0
`LET J = NOT 0`	Asigna a J el valor 1
`LET A = (NOT (7 > 3)) <> 0`	7 es mayor que 3, por lo tanto 7 > 3, se considera 1. NOT 1 es igual a 0. Ya que 0 no es distinto de 0, A tomará por lo tanto el valor 0.



Tablas de verdad

A veces resulta útil recurrir a las llamadas tablas de verdad. Estas no son otra cosa que representaciones esquemáticas del operador lógico, y tienen como objeto proporcionar todos los resultados posibles correspondientes a las distintas combinaciones de la expresión, aclarando y visualizando rápidamente las relaciones entre los valores de entrada y los resultados de salida. La tabla de verdad de NOT es bien sencilla:

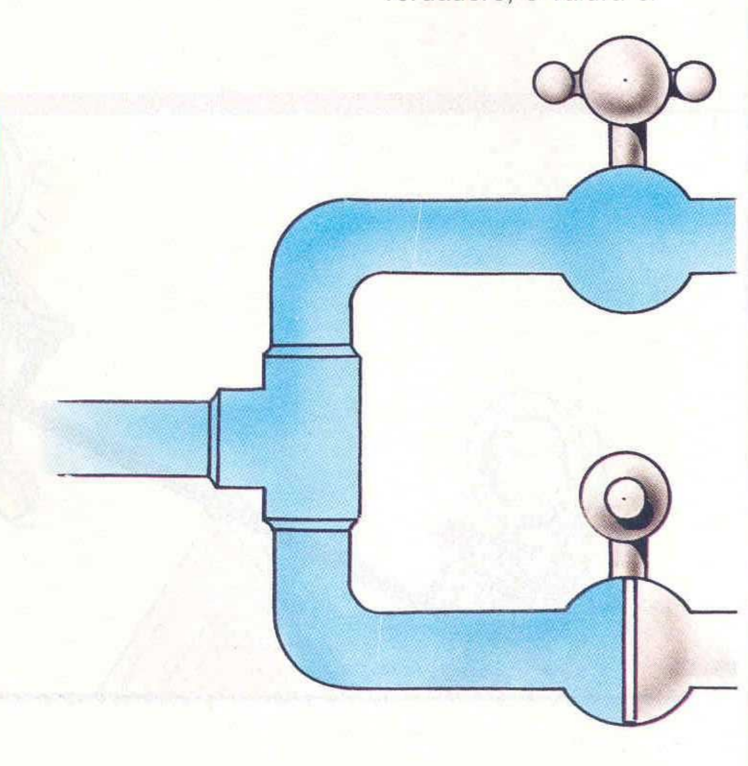
entrada	salida
1	0
0	1

OR

El OR, o suma lógica, se define en cambio de esta manera: «la suma lógica entre dos expresiones lógicas es verdadera (es decir, igual a 1), aunque solamente una de ellas sea verdadera (es decir, 1); si ambas son falsas (iguales a 0), la suma lógica es falsa. Entonces, si definimos la variable C de esta manera:

```
LET C = (A > 1) OR (B > 2)
```

se le asignará el valor 1, si por lo menos una de las dos expresiones entre paréntesis se verifica. Si en cambio ninguno de los dos paréntesis resulta verdadero, C valdrá 0.



Ejemplos:

```
LET I = (G = 3.1) AND (T = 4.5)
```

I tomará el valor 1 siempre que G y T tengan respectivamente valores iguales a 3.1 y 4.5.

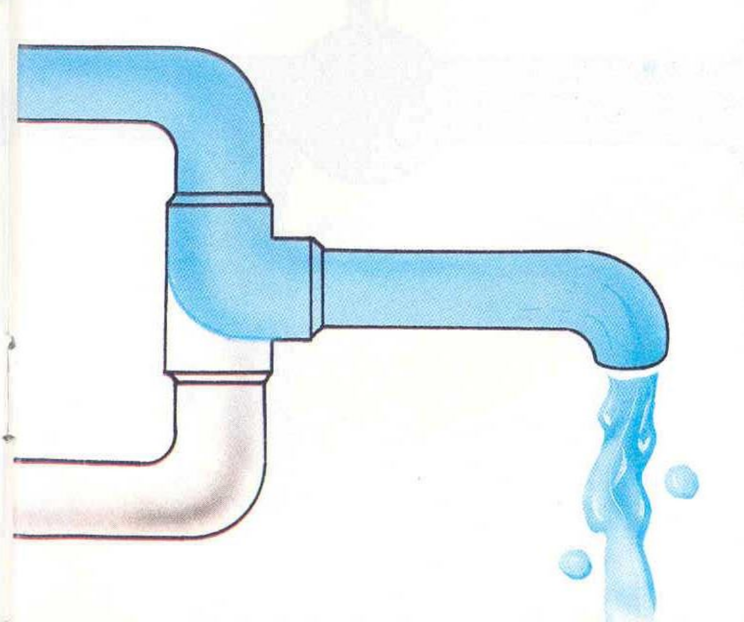
```
LET A = (K > 3 OR J < 1) AND P
```

Para que A tome el valor 1, habrán de resultar: P < > 0, y K > 3 ó bien J < 1.

La tabla de verdad

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

En esta ocasión, la única forma de obtener un resultado verdadero es que ambas expresiones sean verdaderas, es decir, 1. En un cierto sentido se podría decir que el producto lógico es la operación inversa a la suma lógica (comparando las dos tablas AND y OR, lo entenderás fácilmente).

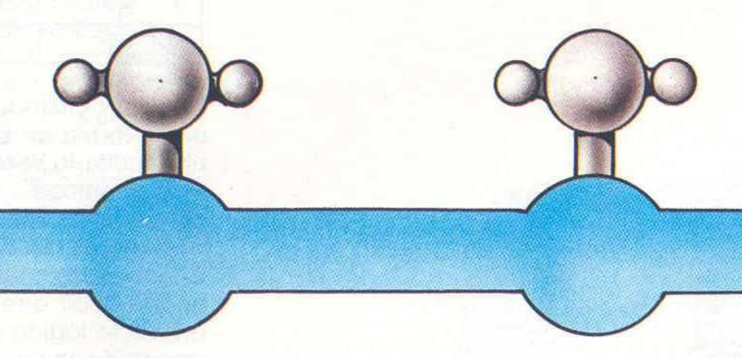


AND

El último operador que nos queda por examinar es AND (o producto lógico).
Se define de la siguiente manera:
«El producto lógico entre dos expresiones lógicas es verdadero (y por lo tanto, igual a 1), si solamente ambas expresiones son verdaderas (iguales a 1).»
La variable F, definida como:

```
LET F = (A > 1) AND (B > 2)
```

resultará por lo tanto, igual a 1 únicamente si ambas expresiones en el interior de los paréntesis son verdaderas, es decir, si A es mayor que 1 y B mayor que 2.
En cambio, aunque uno solo de los paréntesis contenga una expresión falsa, F tomará el valor 0.



Ejemplos:

```
LET J = (5>D) OR (Z = 9)
```

J valdrá 1, si D es menor que 5 o si Z es igual a 9. En el caso contrario valdrá 0.

```
LET M = (NOT A) OR B
```

Si B es distinto de 0 (dado que NOT 0 = 1), a M se le asignará el valor 1. De otro modo M valdrá 0.

La tabla de verdad

La tabla de verdad de OR es la siguiente (A y B son dos expresiones lógicas cualquiera):

A	B	A OR B
0	0	0
0	1	0
1	0	0
1	1	1

Como habrás podido ver, el único caso en que A OR B es igual a 0 es aquél en el que ambas expresiones son falsas, es decir, iguales a cero.



Prioridad de los operadores lógicos

También los operadores lógicos, al igual que los matemáticos, tienen una prioridad.
El orden de las prioridades en las operaciones lógicas, de menor a mayor, es: OR, AND, NOT.
Naturalmente, estas prioridades siempre pueden ser alteradas mediante el empleo de paréntesis; y aún mejor, dado que la vista no está demasiado acostumbrada a leer expresiones como:

```
A = NOT C > A AND B OR D AND C
```

será preferible, para evitar errores o confusiones, emplear siempre los paréntesis:

```
A = (NOT (C > A) AND B) OR (D AND C)
```

En la parte de la lección dedicada a programación encontrarás algunos ejemplos, que te permitirán adquirir práctica y más familiaridad en el uso de los operadores lógicos.

STOP

Imagínate que estés en una sala de montaje sentado delante de la moviola: estás comprobando la secuencia de fotogramas de una película.
De pronto te das cuenta de que falta la secuencia principal (la que explica la «psicología» del protagonista).
Entonces interrumpes la proyección, buscas, y afortunadamente encuentras el trozo que falta, sin el cual la película habría resultado un «rollo», y lo empalmas en el lugar donde debía ir.
Ahora cada cosa está en su sitio y la proyección puede continuar.
Traslada ahora este ejemplo a otro plano: la película es un programa y tú dejas de ser montador y te conviertes en programador.
El programa no funciona perfectamente aunque no has podido encontrar ningún error claro manifiesto; en un momento determinado el programa se bloquea con un mensaje de

error.
Te resultaría verdaderamente cómoda una instrucción que detuviera momentáneamente la ejecución, permitiéndote así comprobar algunos resultados intermedios, y volver a empezar como si no hubiera ocurrido nada...
¿Qué se puede hacer? Para casos como éste el BASIC pone a tu disposición la instrucción STOP.
STOP detiene la ejecución de un programa provocando la aparición de un mensaje que indica en qué línea de programas se produce la interrupción, igual que si se hubiera detectado un error.
La gran ventaja es que

después de un STOP el ordenador pasa al modo «editor»: es decir, puedes insertar líneas nuevas, introducir correcciones, y examinar o modificar el valor de las variables.

Naturalmente, una vez que se hayan resuelto los problemas es necesario eliminar los STOP. A nadie le gustaría que el programa se le parara en medio de la ejecución.
Así, por ejemplo, la línea

```
150 STOP
```

cuando se ejecute visualizará:

```
9 STOP STATEMENT,150
```

avisándote que la interrupción del programa ha tenido lugar en la línea 150, debido a una instrucción de STOP.

Sintaxis de la instrucción

STOP

CONTINUE

Igual que en la moviola, es posible retomar la proyección a partir del punto exacto de la interrupción; así, en un programa que haya sido bloqueado por un STOP, es posible reanudar su ejecución gracias a la orden CONTINUE (que, naturalmente, se imparte en modo inmediato).

STOP y CONTINUE constituyen pues un instrumento utilísimo para el «debugging», es decir, para la corrección y la puesta a punto de los programas.

Gracias a ellos puedes vigilar los puntos estratégicos de un programa, modificando y mejorando si fuera necesario, aquellas instrucciones que en un primer momento habías encontrado correctas, pero sin que esto interrumpa de una forma definitiva la elaboración en curso hasta ese momento.

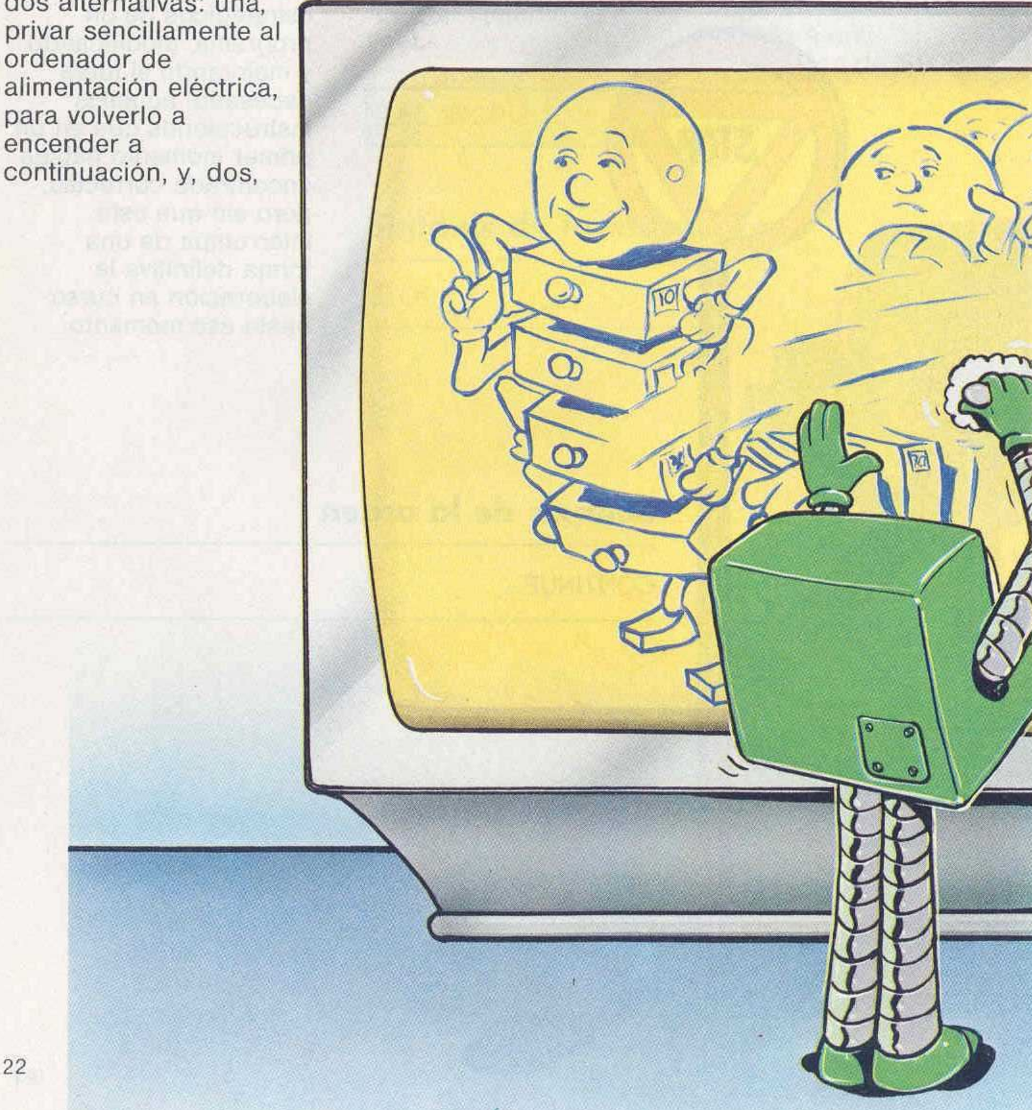


Sintaxis de la orden

CONTINUE

NEW

A veces te encuentras en la necesidad de borrar un programa de la memoria del ordenador.
En este caso existen dos alternativas: una, privar sencillamente al ordenador de alimentación eléctrica, para volverlo a encender a continuación, y, dos, mucho más elegante, hacer NEW.
NEW sirve para borrar el programa BASIC existente en memoria junto a todas las variables. Suele ser empleado cuando se comienza a escribir un programa nuevo, y se quiere tener la seguridad de que no



quede ninguna instrucción, o parte de un programa anteriormente existente en memoria.
Esta orden debe emplearse con sumo cuidado; un NEW indicado sin reflexionar puede provocar en un instante la pérdida de un montón de horas de trabajo.
NEW suele emplearse en modo inmediato, sin embargo también puede ser incluido en el interior de un programa:



```
190 REM CUIDADO: CONTESTANDO NO
    PERDERAS EL PROGRAMA

200 IF A$ = "NO" THEN NEW
```



En este caso, sin embargo, debes tener muy presente que, en el momento de la ejecución de la línea 200, el programa entero se perderá, provocando como consecuencia la interrupción instantánea de toda la actividad del ordenador.
En cualquier caso, será una buena medida que antes de insertar esta orden en un programa aprendas a utilizarla y a comprender su funcionamiento exacto mediante su empleo en modo inmediato, que es el más clásico y seguro.

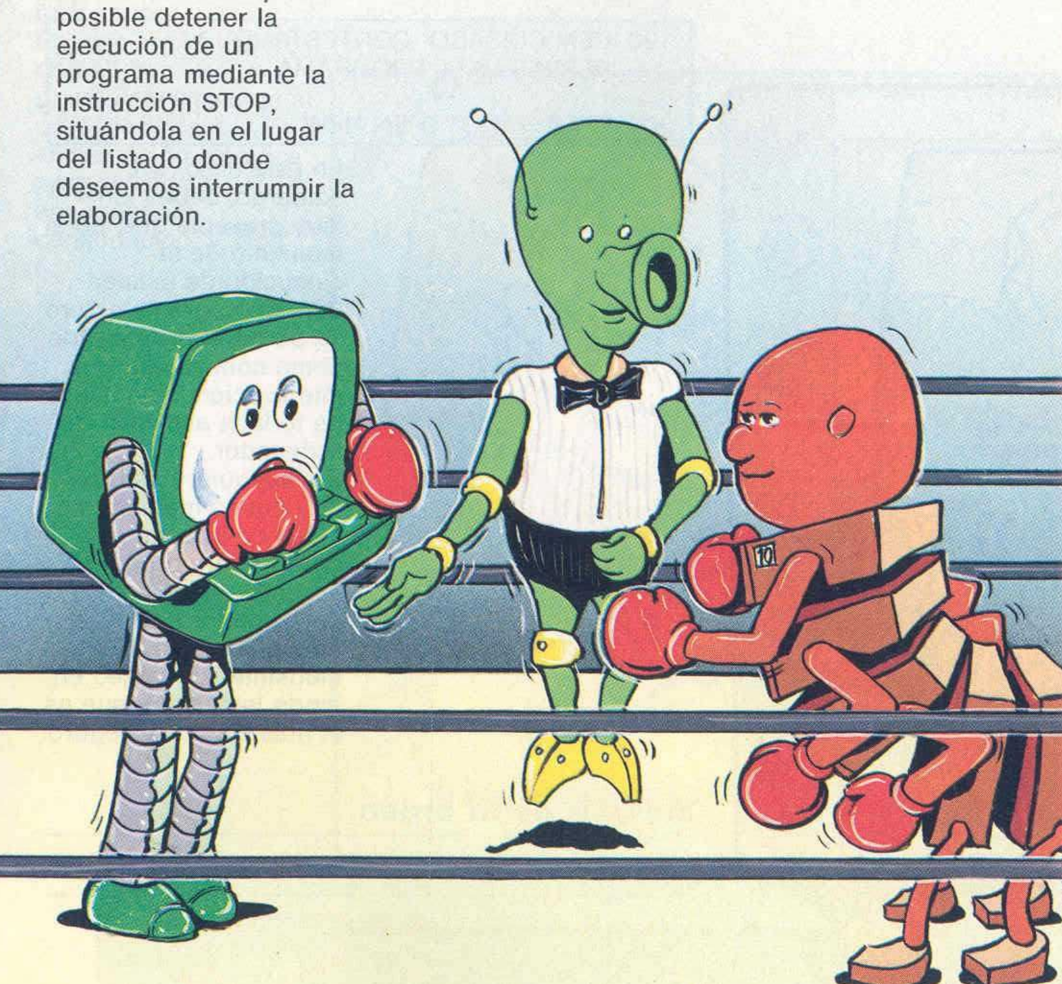
Sintaxis de la orden

NEW

Como interrumpir un programa

Ya hemos visto que es posible detener la ejecución de un programa mediante la instrucción STOP, situándola en el lugar del listado donde deseemos interrumpir la elaboración.

Además, existen casos en los que es necesario interrumpir el problema con una decisión ajena al propio programa: el caso típico es el de un error capaz de conectar un bucle sin fin, del que no se podría salir más que cortando la



alimentación. Pero esta solución tiene la desventaja de borrar todos los datos de la memoria, obligando a empezar de nuevo a recargar el programa. Piensa en una línea como la siguiente:

```
320 GO TO n
```

en la cual n, por cualquier razón, haya tomado el valor 230. Afortunadamente, puedes salir de situaciones como ésta pulsando dos teclas situadas en los extremos de la fila inferior del teclado: CAPS/SHIFT y SPACE. Púlsalas simultáneamente y el programa se detendrá, devolviendo el control del ordenador. Al mismo tiempo aparecerá en pantalla el mensaje:

```
L BREAK
INTO PROGRAM
```

Así, tienes la posibilidad de incluir las correcciones que necesites en el listado, y de retomar la elaboración con un GO TO a la línea modificada.
Además es posible suspender un programa por motivos diferentes: supón que el teléfono suene mientras aparecen los datos que te interesan. Podrías hacer tres cosas:
1) No contestar la llamada.
2) Perder los datos que te interesaban.
3) Hacer BREAK, contestar al teléfono y continur el programa desde el lugar donde lo habías interrumpido.
También puedes emplear BREAK mientras funciona el casete o un periférico como la impresora.
En este caso, al contrario que en los anteriores, es suficiente con pulsar la tecla SPACE; el mensaje que aparece

```
D BREAK - CONT
REPEATS
```

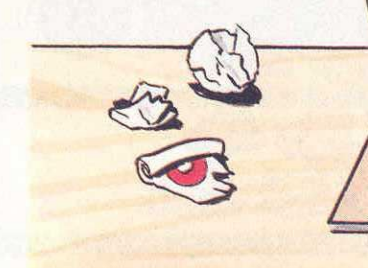
te recuerda que la instrucción CONTINUE hará repetir la instrucción interrumpida en vez de pasar a la siguiente.

CLEAR

CLEAR es una instrucción mucho menos drástica que NEW.
Se limita a borrar de la memoria todas las variables, pero no el programa, que permanece dispuesto para posteriores usos.
Otro efecto de CLEAR es el de limpiar la pantalla, con efecto similar al que produce la instrucción CLS.
Comprueba personalmente lo dicho, introduciendo en tu Spectrum el siguiente programa:

```
10 LET A = 23
20 LET B = 12
30 LET S = A + B
40 PRINT S
50 PRINT "EFECTO CLEAR :"
60 PAUSE 100
70 CLEAR
80 PRINT S
```

Aparecerá el mensaje de error «2 VARIABLE NOT FOUND, 80 : 1», lo que te demuestra que la línea 70 ha borrado las variables.
Pide ahora el listado con LIST y comprueba que el programa permanece en la memoria.
Además, CLEAR se puede emplear con un argumento numérico que indica la última posición de memoria alcanzable por el programa BASIC.
Esto puede ser muy útil



cuando sea necesario reservar una determinada parte de la memoria para datos especiales y se quiera evitar que el programa BASIC durante su redacción los escriba escima, destruyéndolos. La localización de esta posición límite se llama RAMTOP; todas las informaciones memorizadas por encima de ella no serán borradas ni siquiera por NEW.

El número a proporcionar a CLEAR debe quedar comprendido entre el de la última posición física de la memoria (65535) y la primera no ocupada por el programa BASIC y por el Sistema Operativo.

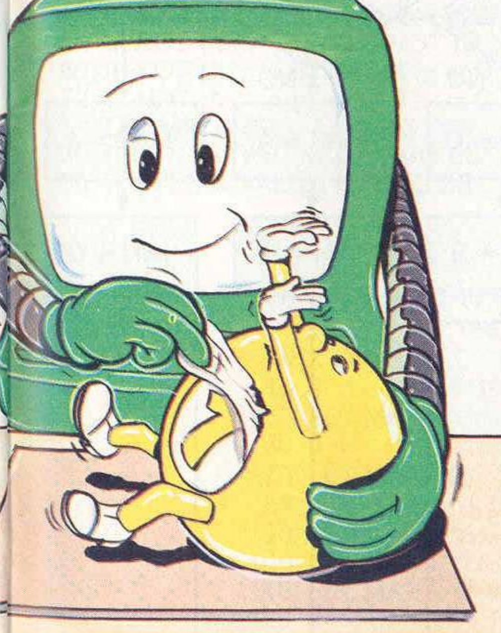
El mensaje de error «RAMTOP NO GOOD»indica la necesidad de incrementar el valor indicado, mientras que «INTERGER OUT OF RANGE» indica que es necesario disminuirlo. Introduce estos ejemplos:

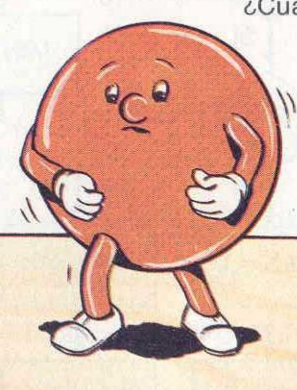
```
10 REM PRUEBA
20 CLEAR 28000
30 POKE 28001, 65
40 REM HE ESCRITO UNA A EN 28001
50 PRINT "AHORA VOYA NEW"
60 PAUSE 200
70 NEW
```

Solicita ahora el listado mediante LIST: no queda ninguna huella del programa. Teclea:

```
PRINT CHR$ (PEEK 28001)
```

¿Cuál es el resultado?

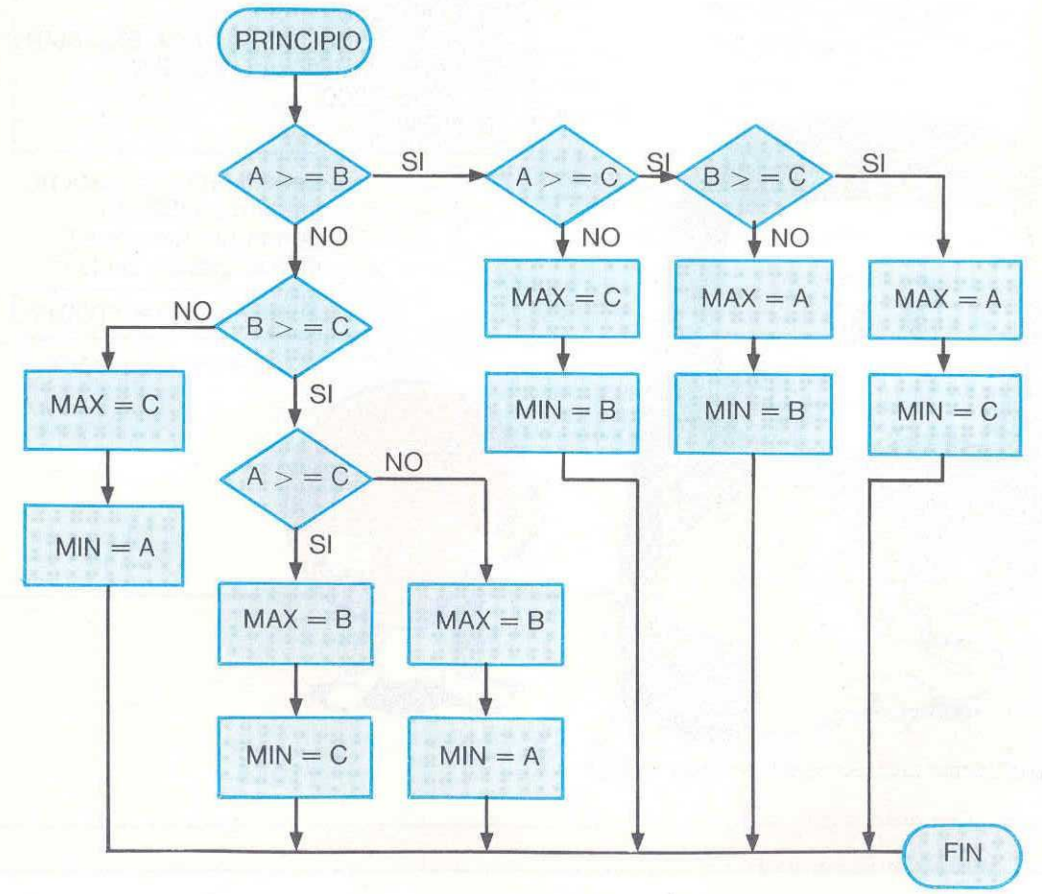




Aplicaciones de los operadores lógicos

Nos proponemos resolver el siguiente problema: dados tres números cualquiera en la entrada, encontrar el mayor y el menor. Suponiendo que llamemos A, B y C a las variables empleadas para insertar en el ordenador los tres valores numéricos, un posible diagrama de flujo podría ser:

La extrema sencillez del problema no requiere ninguna explicación sobre su diagrama de flujo.
En cambio son necesarios algunos comentarios sobre la conversión del diagrama a su correspondiente programa BASIC, del que a continuación encontrarás dos



versiones distintas. Tu ordenador aceptará ambas: desde su punto de vista, mientras que la sintaxis sea correcta, no existe ninguna diferencia entre ellas. Para nosotros, en cambio, es evidente la diferencia de planteamiento, estructura y legibilidad existente entre ambos listados. Gracias al uso intensivo de los operadores lógicos, seguir, comprender, y sobre todo comprobar la función, la ejecución y el desarrollo de cada instrucción de la segunda versión, deberia resultar mucho más sencillo y lineal que hacerlo en la primera.
Esto se debe, en gran parte, a la ausencia de instrumentos GOTO, cuyo fundamental defecto es el de interrumpir la trama lógica seguida en el momento de la realización del diagrama de flujo.
Por lo tanto, te damos un consejo: intenta aprender a programar siguiendo este segundo planteamiento y te convertirás en un excelente programador.

```
10 INPUT A, B, C
20 IF A > = B AND A > = C THEN LET MAX = A: LET MIN = C
30 IF A > = B AND A > = C AND NOT (B > = C) THEN LET MAX = A: LET
   MIN = B
40 IF A > = B AND NOT (A > = C) THEN LET MAX = C: LET MIN = B
50 IF NOT (A < = B) AND NOT (B > = C) THEN LET MAX = C: LET MIN = A
60 IF NOT (A > = B) AND A > = C THEN LET MAX = B: LET MIN = C
70 IF NOT (A > = B) AND B > = C AND NOT (A > = C) THEN LET MAX = B:
   LET MIN = A
80 PRINT "EL VALOR MAXIMO ES"; MAX
90 PRINT "EL VALOR MINIMO ES"; MIN
```

```
 10 INPUT A, B, C
 20 IF A > = B THEN GOTO 40
 30 GOTO 80
 40 IF A > = C THEN GOTO 60
 50 LET MAX = C: LET MIN = B: GOTO 120
 60 IF B > = C THEN LET MIN = C: LET MAX = A: GOTO 120
 70 LET MIN = B: LET MAX = A: GOTO 120
 80 IF B > = C THEN GOTO 100
 90 LET MAX = C: LET MIN = A: GOTO 120
100 IF A > = C THEN LET MAX = B: LET MIN = C: GOTO 120
110 LET MAX = B: LET MIN = A
120 IF MAX = MIN THEN PRINT "VALORES IGUALES": STOP
130 PRINT "EL VALOR MAXIMO ES"; MAX
140 PRINT "EL VALOR MINIMO ES"; MIN
```

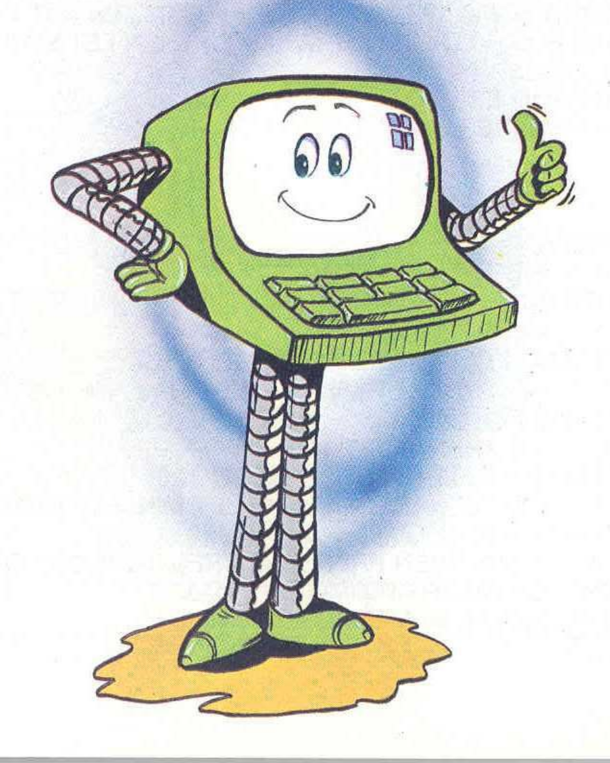
Ahora, una última obsevación, esta vez referente a la sintaxis. ¿Qué ocurriría si por ejemplo en la línea

```
30 IF A > = B AND A > = C AND NOT (B > = C) THEN LET MAX = A: LET
   MIN = B
```

no se pusieran los paréntesis?

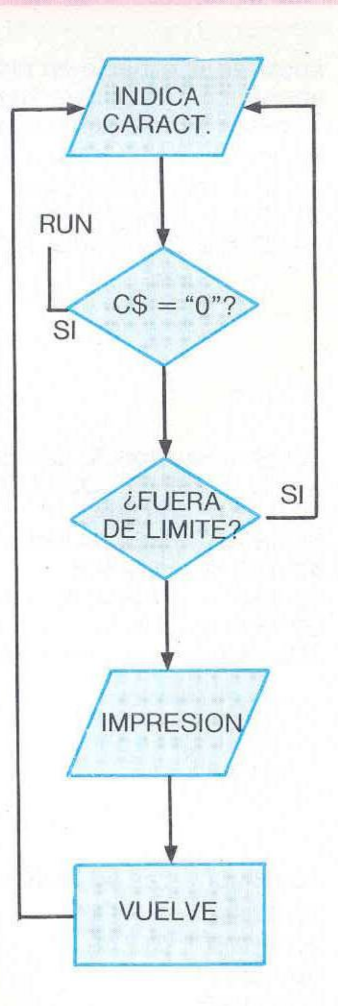
```
30 IF A > = B AND A > = C AND NOT B > = C THEN LET MAX = A: LET
   MIN = B
```

¡Piénsalo bien! Es importante la prioridad de ejecución de las operaciones lógicas.



El juego de la palabra

Se trata de un sencillo juego para dos o más jugadores: la tarea de cada uno de ellos es introducir por turno a través del teclado una letra del alfabeto (de A a la Z).
Un posible reglamento le asignaría la victoria al jugador que al añadir una letra formara una palabra con pleno sentido, o por el contrario, declarar derrotado a este jugador.
El programa emplea el operador lógico OR para comprobar la validez del carácter introducido, y por la misma razón usa INKEY$ en lugar de INPUT.
Además es muy importante el signo de puntuación; después de la variable en la línea 40, este permite que la posición de impresión permanezca invariable (para darte cuenta de su importancia, elimínalo).




```
10 PAUSE 0 : LET C$ = INKEY$
20 IF C$ = "0" THEN RUN
30 IF C$ < "A" OR C$ > "Z" THEN GOTO 10
40 PRINT C$;
50 GO TO 10
```

EJERCICIOS

Anota en el espacio en blanco el resultado que preveas para cada ejercicio propuesto, y después cotéjalo con la solución de tu ordenador. Si has cometido un solo error, tendrás que repasar la lección.

```
10 LET A = 12: LET B = 5: LET N$ = "BIEN"
20 IF A < B THEN PRINT "MANZANAS"
30 IF B < A THEN PRINT "PLATANOS"
40 IF (A + B) <> (B + A) THEN PRINT N$
```

¿Cuál es el aspecto que consideras más importante de este listado?

```
10 CLS: PRINT "¿QUE NOTA MERECES?"
20 INPUT "TEORIA"; T
30 INPUT "PROGRAMACION"; P
40 LET M = T + P/2
50 IF P > = 7 AND M > 6 THEN PRINT "APROBADO"
60 IF P < 7 OR M < = 6 THEN PRINT "SUSPENDIDO"
```

¿Crees que con 3 en teoría y 8 en programación obtendrás un aprobado?

¿Cuántas veces se realizará la elaboración de este ciclo?

```
10 FOR I = 1 TO 10
20 PRINT I: CLEAR
30 NEXT I
```

¿Por qué?

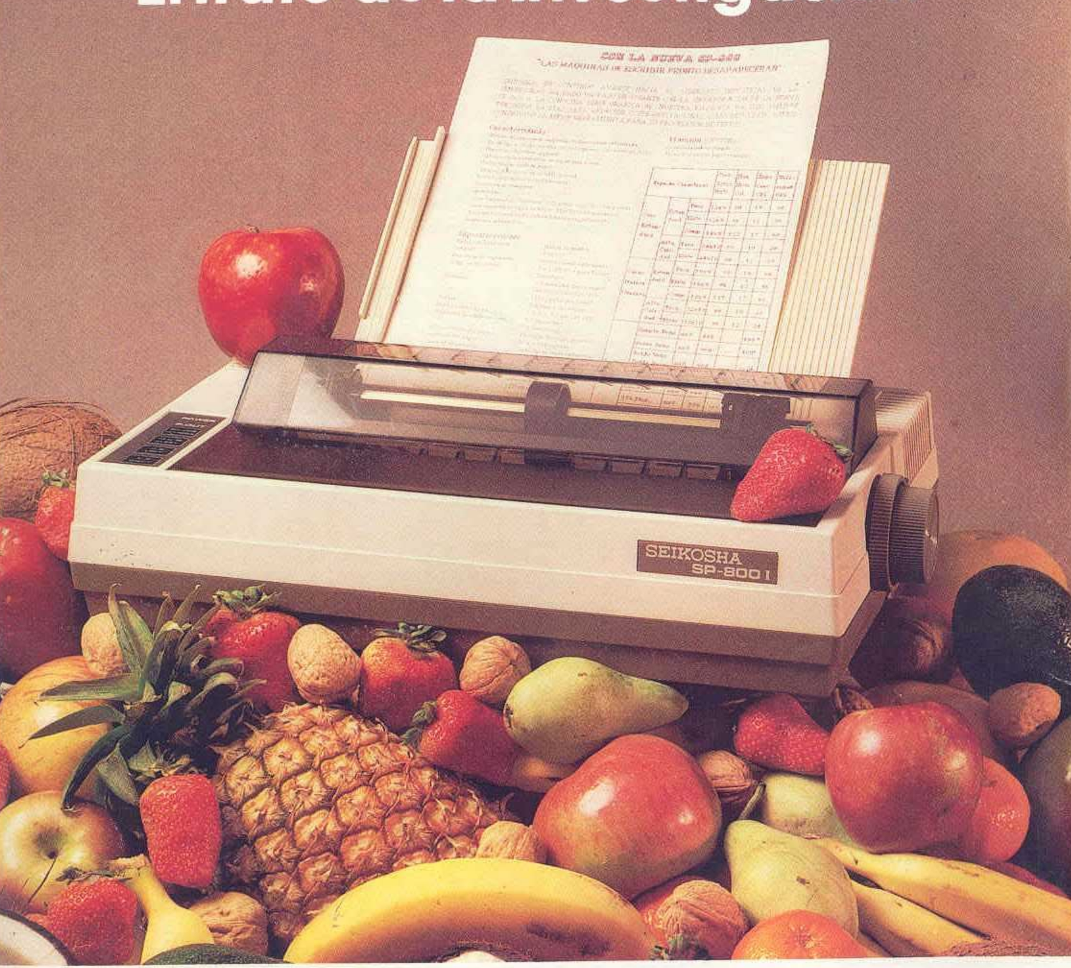
Para los amantes de esconder sus secretos...

```
10 PAPER 2: INK 2: CLS
20 PRINT AT 4,4; "ES UN SECRETO"
30 PAUSE 100
40 INK 7: PRINT AT 10,4; "NO ES UN SECRETO:
```

Una técnica empleada para ocultar los listados de ojos indiscretos es la de imprimirlos con una tinta igual al color del fondo (papel).

SEIKOSHA SP-800

El fruto de la Investigación



La nueva impresora de SEIKOSHA SP-800, con un ordenador personal puede escribir ***96 combinaciones de letra diferentes***, desde 96 caracteres por segundo a 20 con muy alta calidad de letra, además es gráfica en alta densidad.

Su precio es de 69.900 Pts con ***introductor automático hoja a hoja.***

Con un pequeño ordenador personal, un procesador de textos puede costar alrededor de cien mil pesetas.

Infórmese y comprenderá por qué **las máquinas de escribir** tienen demasiados años.

Nuestra calidad es "SEIKO";
nuestros precios, únicos
Si desea más información,
consulte con nuestro distribuidor
más cercano, llame o escriba a:

DIRAC

DIRECCION COMERCIAL:
Av. Blasco Ibañez, 114-116
46022 VALENCIA
Tel (96) 372 88 89
Télex 62220
DIRECCION COMERCIAL EN CATALUÑA:
C/Muntaner, 60-2-4Pta
08011 BARCELONA
Tel.(93) 323 32 19

ESTOS SON NUESTROS MODELOS:

MODELO	VELOCIDAD	COLUMNAS	TIPOS DE LETRA	P.V.P.R. INTERFACE PARALELO
GP-50S LA DEL SPECTRUM	40 cps	32	-	19.900
GP-50 LA PEQUEÑA	40 cps	46	2	25.900
GP-500 LA ECONOMICA	60 cps	80	2	47.900
GP-700 LA DE COLOR	50 cps	80-106	3	69.900
SP-800 LA PERFECCION	96 cps	80-137	20	69.900
BP-5200 LA DE OFICINA	200 cps	136-272	18	199.900
BP-5420 LA MAS RAPIDA	420 cps	136-272	18	299.900

* Los precios indicados son los recomendados para conexion tipo paralelo Centronics, para otro tipo de conexion, sufren un ligero incremento.

Este pie de página ha sido realizado integramente con la nueva impresora:

SEIKOSHA SP-800